National University of Singapore
School of Computing
CS1010S: Programming Methodology
Semester I, 2018/2019

Solutions for Recitation 7
Multiple Representations

Problems

1. **Dense Matrix Representation**. A matrix can be represented in Python by a list of lists (nested lists). For example, `m = [[ 1, 2, 3], [4, 5, 6], [7, 8, 9]]` represents the following 3×3 matrix:

$$\begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix}$$

   You are given the following implementation for `make_matrix(seq)`, which takes in a sequence, i.e. either a tuple or a list, and creates the matrix object.

```
def make_matrix(seq):
    mat = []
    for row in seq:
        mat.append(list(row))
    return mat
```

   (a) Suppose `seq` were a list of lists. Would the following implementation of `make_matrix(seq)` work? Explain.

```
def make_matrix(seq):
    return seq
```

   No, this would not work because it would result in aliasing, i.e. if `seq` were changed, then the matrix object would be affected. Typically, we need to make a copy of the state when we deal with mutable representations.

   (b) Implement the following supporting functions:

   i. `rows(m)`: returns the number of rows for matrix object `m`.

```
def rows(mat):
    return len(mat)
```

   ii. `cols(m)`: returns the number of columns for matrix object `m`.

```
def cols(mat):
    return len(mat[0])
```

   iii. `get(m,x,y)`: returns the element (i,j) for matrix object `m`.

```
def get(mat,x,y):
    return mat[x][y]
```

iv. `set(mat,x,y,val)`: sets the element (i,j) for matrix object `m` to value `val`.

```
def set(mat,x,y,val):
    mat[x][y] = val
```

v. `transpose(m)`: transposes matrix object `m`. Basically, this converts a $m \times n$ matrix into a $n \times m$ matrix.

```
def transpose(mat):
    transposed = []
    for i in range(len(mat[0])):
        column = []
        for j in range(len(mat)):
            column.append(mat[j][i])
        transposed.append(column)
    mat.clear()
    mat.extend(transposed)
```

If we were to replace this snippet of code:

```
    mat.clear()
    mat.extend(transposed)
```

with

```
    mat = transposed
```

this code will fail because of aliasing.

vi. `print_matrix(mat)`: prints the contents of matrix object `m` in a human readable form.

```
def print_matrix(mat):
    for row in mat:
        print(row)
```

2. **Sparse Matrix Representation**. Now suppose that implementation of `make_matrix(seq)` is as follows:

```
def make_matrix(seq):
    data = []
    for i in range(len(seq)):
        for j in range(len(seq[0])):
            if seq[i][j] != 0:
                data.append([i,j,seq[i][j]])
    return [len(seq),len(seq[0]),data]
```

(a) Implement the list of associated functions listed in Part 1(ii) above.

i. `rows(m)`

```
def rows(mat):
    return mat[0]
```

ii. `cols(m)`

```
def cols(mat):
    return mat[1]
```

iii. `get(m,x,y)`

```
def get(mat,x,y):
    for record in mat[2]:
        if record[0] == x and record[1] == y:
            return record[2]
    return 0
```

Alternatively, we can also do:

```
def get(mat,x,y):
    for i,j,val in mat[2]:
        if i == x and j == y:
            return val
    return 0
```

iv. `set(mat,x,y,val)`

```
def set(mat,x,y,val):
    for record in mat[2]:
        if record[0] == x and record[1] == y:
            record[2]=val
            return
    mat[2].append([x,y,val])
```

Note that this is a naive solution. It is possible for `val` to be zero and for there to be a non-zero initial value for `(x,y)`. In such event, we should remove the corresponding record. This is left as an exercise for the reader.

v. `transpose(m)`

```
def transpose(mat):
    for record in mat[2]:
       record[0],record[1]=record[1],record[0]
    mat[0],mat[1]=mat[1],mat[0]
```

vi. `print_matrix(mat)`

```
def print_matrix(mat):
    temp = []
    zeros = [0]*mat[1]
    for row in range(mat[0]):
        temp.append(list(zeros))
    for record in mat[2]:
        temp[record[0]][record[1]] = record[2]
    for row in temp:
        print(row)
```

(b) Which is the better implementation for the matrix object? Explain.

There is no "better" representation. There are different tradeoffs. The better representation will depend on: (i) the actual data; and (ii) how the data structures are used.